A Lightweight and Scalable Attribute-Based Encryption System for Smart Cities

Marco Rasori[a,1,*], Pericle Perazzo[a], Gianluca Dini[a]

[a]*Department of Information Engineering, University of Pisa, Pisa, Italy*

Abstract

In the near future, a technological revolution will involve our cities, where a variety of smart services based on the Internet of Things will be developed to facilitate the needs of the citizens. Sensing devices are already being deployed in urban environments, and they will generate huge amounts of data. Such data is typically outsourced to some cloud service in order to lower capital and operating expenses and guarantee high availability. However, cloud services may suffer from data breaches due to software and hardware vulnerabilities, or they may have incentives to release stored data to unauthorized entities. In this work we present ABE-Cities, an encryption system for urban sensing which solves the above problems while ensuring fine-grained access control on data by means of Attribute-Based Encryption (ABE). ABE-Cities senses data from the city and stores it on the cloud in an encrypted form. Then, it provides users with keys able to decrypt only data sensed from authorized paths or zones of the city. In ABE-Cities, sensors perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices such as battery powered motes. ABE-Cities allows us to plan an expiration date for each key, as well as to revoke a given key in an unplanned fashion. We prove that ABE-Cities scales well with the number of users and the number of streets by simulating it with 30 000 users on the Beijing street network, which consists of more than 30 000 streets. In addition to the "vanilla" ABE-Cities scheme, we propose an "advanced" one that leverages the presence of IoT gateways to reduce the computational load otherwise weighing on a single Trusted Third Party. We validate this by testing the advanced scheme on the simulated Houston and Beijing street networks.

Keywords: Smart City, Urban Sensing, Attribute-Based Encryption, Internet of Things, Data-Centric Security

*Corresponding author
Email addresses: `marco.rasori@ing.unipi.it` (Marco Rasori), `pericle.perazzo@iet.unipi.it` (Pericle Perazzo), `gianluca.dini@iet.unipi.it` (Gianluca Dini)
[1]Department of Information Engineering, University of Florence, Florence, Italy

1. Introduction

A *smart city* is an urban environment that offers digital services to improve the life quality of the citizens. Such services span across all sectors of society including health, logistics, and mobility, and they often capitalize on the Internet of Things (IoT) as enabling technology. Smart devices underlying these services produce large amounts of data which can be outsourced to a *cloud server*. This is usually preferred to an in-house solution because it reduces costs while providing high availability. In many cases, sensed data includes sensitive or valuable information which is intended to be read only by authorized users. Unfortunately, as soon as data lands on the cloud, we lose any control on it and we have to completely trust the cloud server. In particular, cloud servers are highly exposed to a plethora of attacks: from "classic" injection [1] to recent hardware-based attacks like Spectre [2] and Meltdown [3]. A data breach on a smart city system could either represent an important loss of money for the data owner or a serious privacy violation for the citizens[2]. Moreover, cloud service providers may have some incentive to release stored information to others [4, 5].

A possible solution to this problem is to store data in an encrypted form on the cloud server. This can be done by means of *Attribute-Based Encryption* (ABE) [6, 7, 8, 9], which also ensures a fine-grained access control over data. In ABE everyone can encrypt because only public parameters are used for encryption. Decryption is instead performed by means of a *decryption key*, which is specific to each user and generated by a *Trusted Third Party* (TTP). Decryption keys and encrypted data "embed" *attributes* and *access policies*, which are basically Boolean formulas defined over the attributes. A given decryption key is able to decrypt a given piece of data only if the embedded *access policy* is satisfied by means of the embedded attributes.

In this paper we present ABE-Cities, an encryption system for smart city applications employing Attribute-Based Encryption. ABE-Cities allows fine-grained access control over the encrypted data stored in the cloud server, and it is secure against traffic eavesdropping, sensors compromise, and data leakage from the cloud server. ABE-Cities senses data from the city and stores it on the cloud server in an encrypted form. Then, it provides users with keys able to decrypt only data sensed from authorized paths or zones of the city. In ABE-Cities, sensors perform only lightweight symmetric-key encryption, thus we can employ resource constrained sensor devices such as battery powered motes. This makes ABE-Cities suitable for a broad set of IoT smart city applications. ABE-Cities allows us to plan an expiration date for each key, as well as to revoke a given key in an unplanned fashion. We prove that ABE-Cities scales well with the number of users and the city size by simulating it with 3000 users on the Houston street network (12 000+ streets), and with 30 000 users

[2]https://securityintelligence.com/ponemon-cost-of-a-data-breach-2018/

on the Beijing street network (30 000+ streets). In addition to the “vanilla” ABE-Cities scheme, we propose an “advanced” one that leverages the possible presence of IoT gateways to reduce the TTP computational load. This advanced version is based on Merkle trees, and it takes advantage of nodes acting as sensor gateways to outsource a significant amount of computation from the TTP. We finally validate the advantages introduced by the advanced scheme by testing the encryption performance on Houston and Beijing street networks. We remark that with respect to generic IoT ABE systems (e.g., [10]), ABE-Cities is particularly suitable for smart city applications. This is because ABE-Cities uses a street network representation that makes key revocations efficient, and it avoids broadcast communications which may be infeasible in sensor networks distributed over large urban areas.

This paper extends our previously published conference paper [11]. As a new contribution with respect to the conference paper:

- we extensively tested the scalability of ABE-Cities by simulating it on large street networks, namely Houston (12 000+ streets) and Beijing (30 000+ streets);
- we developed the advanced version of ABE-Cities which reduces the TTP computational load by outsourcing a significant amount of computation to IoT gateways;
- we showed the advantages introduced by the advanced scheme by testing the encryption performance on Houston and Beijing street networks.

The paper is organized as follows. Section 2 compares with relevant related work. Section 3 introduces some background on ABE and other techniques that we use in our system. Section 4 describes the vanilla ABE-Cities scheme, and the adversary model. Section 5 introduces different attribute representations of the street network, resulting in different system performance. Section 6 compares quantitatively the different attribute representations by simulations, and it evaluates the scalability of ABE-Cities on small and large street networks. Section 7 describes the ABE-Cities advanced scheme. Section 8 validates the advanced scheme by testing its encryption performance on simulated large street networks. Section 9 summarizes our results and concludes the paper.

2. Related Work

ABE has been used to protect the confidentiality in IoT [12, 10, 13, 14, 15, 16, 17, 18, 19], digital health [20, 21, 22, 23], military battlefields [24], and online social networks [25].

Yu et al. [26] first used ABE techniques for outsourcing sensitive data to a semi-trusted cloud storage, while enforcing fine-grained access control at the same time. Their scheme is well applicable in digital health applications, where patients own privacy-sensitive records and can employ full computational capabilities. However, it is less suitable for IoT applications, in which data is produced by constrained devices, which usually lack the necessary computational power and energy to perform ABE encryption. In our system, IoT devices perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices and battery-powered motes. ABE-Cities is thus suitable for a broader set of IoT applications.

Wang et al. [27] proposed a hierarchical ABE scheme that allows the TTP to delegate part of the responsibility to other authorities, which can independently make decisions on the semantics of their attributes. Hierarchical ABE allows also for proxy re-encryption, but it forces a fixed structure for the access policies, so it limits the flexibility of the access control system. In this paper, we focused on a non-hierarchical ABE scheme with a single TTP, leaving possible hierarchical extensions for future work.

Yu et al. [10] proposed an ABE scheme to encrypt data sensed by a wireless sensor network (WSN). The authors used broadcast encryption to revoke keys in an efficient manner. Although their system is suitable for a large set of application scenarios involving locally distributed WSNs, it could be unpractical for geographically distributed ones, like those employed in a smart city. This is because an actual broadcast channel could not be realizable in these scenarios. In our system, we use a key revocation method whose efficiency does not depend on the presence of a broadcast channel, but rather on a convenient attribute representation of the smart city. Moreover, the authors of [10] perform ABE encryption directly on the sensors, which could not have the necessary computational power and energy to do it. In our system the IoT devices perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices and battery powered motes.

Yao et al. [12] proposed an ABE scheme based on elliptic curves instead of pairings. This makes ABE operations more lightweight and thus more suitable for IoT constrained devices. In our scheme this feature is not needed, since the IoT devices perform only lightweight symmetric-key encryption. This permits us to use the pairing-based ABE scheme of Goyal et al. [7], which allows us to employ advanced features unavailable in the Yao et al.'s scheme, like proxy re-encryption [26].

Recently, Odelu et al. [13] proposed an ABE scheme for IoT applications, which guarantees $\mathcal{O}(1)$ encryption and decryption times. Such a scheme forces the access policies to use only AND operators, so it limits the access control flexibility. On the contrary, our system is based on Goyal et al's scheme, which offers a far greater degree of expressiveness.

3. Preliminaries

3.1. Key-Policy Attribute-Based Encryption

Key-Policy ABE (KP-ABE) is a recent public-key encryption paradigm introduced by Goyal et al. [7] in 2006. In KP-ABE an encrypting party labels each piece of encrypted data (*ABE ciphertext*) with a set of attributes which describes it (*encryption attributes*, γ). Decryption keys embed an access policy

($\mathcal{T}$), which is basically a Boolean formula defined over some attributes (*access policy attributes*, λ). An access policy can be represented as a tree in which leaves are access policy attributes, and non-leaf nodes are Boolean operators. A given decryption key is able to decrypt a given ABE ciphertext only if the access policy can be satisfied by using the attributes labeling the ciphertext.

The set of all the attributes used in a given KP-ABE scheme is called *universe of the attributes* ($\mathcal{U}$). Without losing in generality, in the following we will indicate an attribute in the universe either with its unique name, or with a unique natural number, which is more convenient in formulas. Therefore, the attribute sets $\mathcal{U}$, γ, and λ are subsets of $\mathbb{N}$. Who encrypts data is called a *data producer*, whereas who holds a decryption key and decrypts data is called a *data consumer*.

In this paper we build on the original KP-ABE scheme introduced by Goyal et al. [7]. In order to ease the reading, we abstract away the mathematical insight of such a scheme. The interested reader can refer to [7] for such details. We model the Goyal et al.'s scheme with the following black-box primitives.

$(MK, PK) = \text{Setup}(\mathcal{U})$. This primitive initializes the scheme. It takes as input the universe of the attributes $\mathcal{U}$ and generates a random *master key* $MK = (y, t_{i\in\mathcal{U}})$, which must be kept secret by the TTP, and an associated set of *public parameters* $PK = (Y, T_{i\in\mathcal{U}})$, which must be distributed to the data producers. Each attribute i in the universe is associated to a t_i and a T_i. The Setup primitive is executed by the TTP.

$E = \text{Encrypt}(M, \gamma, Y, T_{i\in\gamma})$. This primitive encrypts a message M with the encryption attributes γ. It takes as input Y and $T_{i\in\gamma}$, which are public parameters, and it produces the ABE ciphertext $E = (\gamma, \tilde{e}, e_{i\in\gamma})$. Each encryption attribute $i \in \gamma$ is associated to an *ABE ciphertext component* e_i. The Encrypt primitive is executed by a data producer.

$DK = \text{KeyGen}(MK, \mathcal{T})$. This primitive generates a decryption key $DK = (\mathcal{T}, \lambda, dk_{i\in\lambda})$, which is provided to a data consumer. It takes as input the master key MK and an access policy $\mathcal{T}$, with access policy attributes λ. Each access policy attribute $i \in \lambda$ is associated to a *decryption key component* dk_i. The KeyGen primitive is executed by the TTP.

$M = \text{Decrypt}(E, DK)$. This primitive retrieves the message M from the ABE ciphertext E if the decryption key DK is *authorized* to decrypt, i.e., if the access policy $\mathcal{T}$ embedded in the decryption key is satisfied by means of the encryption attributes γ labeling the ABE ciphertext. It produces $\perp$ otherwise. The Decrypt primitive is executed by a data consumer.

3.2. Proxy Re-Encryption

Proxy Re-Encryption (PRE) is a technique which allows an entity, given an encrypted message, to produce the same message encrypted with a different key, without accessing the message itself. By using this technique, one can re-encrypt old ciphertexts in order to prevent a revoked key to decrypt them and outsource this burdensome operation to a cloud server without compromising data confidentiality.

In this paper we build on the Yu et al. PRE scheme [26], which implements the revocation of a decryption key by means of a system-wide update of a subset μ of the access policy attributes ($\mu \subseteq \lambda$) to a new *version*. The subset μ is called *updatee set* and contains the minimal set of attributes without which the decryption key cannot ever be satisfied. As a consequence, after the system-wide update, the revoked decryption key will not be able to decrypt any ABE ciphertext anymore. Updating an attribute i to a new version means that all the cryptographic quantities in the system related to that attribute are changed. Specifically, for each attribute $i \in \mu$, (1) the t_i of the master key, (2) the T_i of the public parameters, (3) the e_i of all the ABE ciphertexts having i as encryption attribute, and (4) the dk_i of all the decryption keys (except the revoked one) having i as access policy attribute are updated to new quantities. We indicate such new quantities respectively with t'_i, T'_i, e'_i, and dk'_i. The t'_i's and T'_i's are computed by the TTP. On the other hand, the computation of the e'_i's and the dk'_i's is outsourced to the cloud server. To do this, for each attribute $i \in \mu$ the TTP computes a *re-encryption key* (rk_i) and gives it to the cloud server, which applies it to each e_i to obtain an e'_i, and to each dk_i to obtain a dk'_i.

Note that the computation of all the dk'_i's cannot be completely outsourced to the cloud server, otherwise it would know all the decryption keys, and thus a data breach would compromise the entire system. To avoid this, the cloud server is provided with all the decryption key components except for those relative to a special attribute (*dummy attribute*) which has no meaning and is never updated to a new version. The dummy attribute is ANDed at the root of the access policy of every decryption key, and it is included as an encryption attribute in all the ABE ciphertexts. In this way, a decryption key cannot decrypt anything without the component relative to the dummy attribute, which is known only by the data consumer holding the whole decryption key. A data breach on the cloud server would leak only incomplete decryption keys, which are useless.

Proxy re-encryption is performed in a lazy fashion (*lazy re-encryption*), meaning that the cloud server does not perform any re-encryption operation at the moment of the key revocation, but only afterwards when needed. At the time of a revocation, the TTP produces a re-encryption key for each attribute in the updatee set and adds it to a *re-encryption key history* (*RKH*) stored in the cloud server. RKH_i is a table that keeps track of all the re-encryption keys relative to the attribute i, one for each version update of the attribute ($rk_i, rk'_i, rk''_i, \dots$). The cloud server updates components only when a data consumer asks the cloud server for an ABE ciphertext. If either an ABE ciphertext component or a consumer's decryption key component is outdated of more than one version, then the cloud server updates it by using more than one re-encryption key from the history. Finally, the cloud server provides the data consumer with the ABE ciphertext (s)he asked for and, possibly, with the updated decryption key components.

In order to ease the reading, we abstract away the mathematical insight of the Yu et al.'s PRE scheme. The interested reader can refer to [26] for such details. We model the PRE scheme with the following black-box primitives.

$(t'_i, T'_i, rk_i) = \text{UpdateAttribute}(i, MK)$. This primitive updates the attribute i to a new version. It takes as input the master key MK, and it produces the new quantities t'_i, T'_i, and the re-encryption key rk_i. The UpdateAttribute primitive is executed by the TTP.

$e'_i = \text{UpdateE}(i, e_i, rk_i)$. This primitive updates an ABE ciphertext component e_i to a new version e'_i by means of the re-encryption key rk_i. The UpdateE primitive is executed by the cloud server.

$dk'_i = \text{UpdateDK}(i, dk_i, rk_i)$. This primitive updates a decryption key component dk_i to a new version dk'_i by means of the re-encryption key rk_i. The UpdateDK primitive is executed by the cloud server.

3.3. Segment Trees

Segment trees [28] are data structures useful in ABE schemes to implement efficient point-in-interval access policies [29]. A segment tree represents a set of intervals in such a way that it is efficient to query which intervals contain a given number (or *point*). Although segment trees can in principle represent any point and interval in $\mathbb{R}$, in this paper we focus on segment trees capable of representing discrete points and discrete intervals included in a limited number range $[1, \rho]$, with $\rho \in \mathbb{N}$. In the following, we will generically use the term "segment tree" to intend such a specific type of segment tree.

A segment tree over the range $[1, \rho]$ is a binary tree in which each point in $[1, \rho]$ is relative to a leaf.

A generic point ν in the range $[1, \rho]$ is represented by a *point representation set* RS_ν, which is formed by the nodes on the path from the root to the leaf relative to the point. It can be shown that every point representation set is $\mathcal{O}(\log \rho)$ nodes. On the other hand, a generic interval ι included in the range is represented by an *interval representation set* RS_ι, which is the minimum set of nodes whose descendant leaves form the interval. It can be shown that every interval representation set is $\mathcal{O}(\log \rho)$ nodes. By construction, point ν belongs to interval ι iff RS_ν and RS_ι have a non-empty intersection, i.e., $\nu \in \iota \Leftrightarrow RS_\nu \cap RS_\iota \neq \varnothing$.

Segment trees are useful in ABE schemes to implement efficient point-in-interval access policies [29], i.e., access policies which are satisfied iff a given point ν belongs to a given interval ι. According to the terminology in [29], we consider only "KP-ABE Type 1 construction", which means that the ciphertext embeds the point, and the access policy embeds the interval. The point-in-interval access policy is thus implemented in the following way. Each node of the segment tree is represented by an attribute. The ABE ciphertext is labeled with the attributes representing the nodes of the point representation set RS_ν. The access policy is an OR operator between the nodes representing the interval representation set RS_ι. By construction, the access policy is satisfied iff RS_ν and RS_ι have non-empty intersection, that is, iff ν is in ι.

4. ABE-Cities

In ABE-Cities, a city is represented by a *street network*, i.e., a graph in which the edges represent *road segments.* Each street is composed of one or more road segments, and it has a unique identifier. A *sensor* is a constrained device placed on a road segment that acquires data (*sensed data*) relative to such a road segment. The sensed data is stored in an encrypted form in the cloud server, where it is made read-only accessible to the *users.* Users represent the data consumers in our system. Each user is authorized to access data sensed from some paths or zones of the city within a specific time period. For example, supposing the sensors to be IP cameras for traffic monitoring, the user can be authorized to access videos showing the route (s)he usually travels from home to work and back. In this way (s)he can detect traffic congestion in advance and possibly choose different routes. To do this, IP cameras could store in the cloud server multiple short video files in an encrypted form, in such a way to implement an encrypted streaming. The user may also monitor multiple routes in order to identify the least congested one day by day. Moreover, some high privileged user could monitor all the city, for example for providing a transportation management service.

Figure 1 shows the architecture of ABE-Cities. In the following, we will

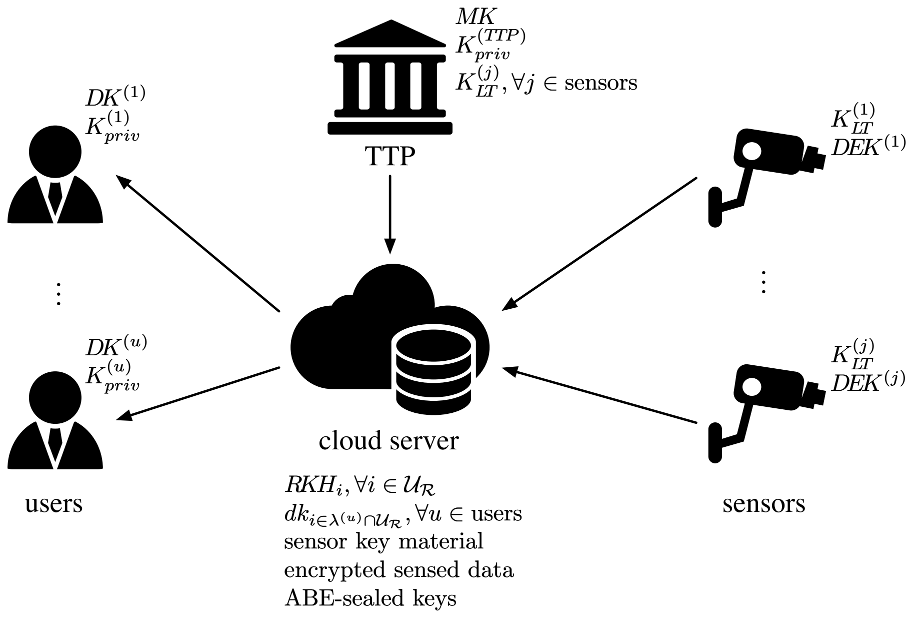


Figure 1: ABE-Cities architecture.

give a general and intuitive description of the system, and in Section 4.1 we will describe the system procedures in detail. The TTP holds the master key and a public/private key pair $(K_{pub}^{(TTP)}, K_{priv}^{(TTP)})$ through which it can perform digital signature algorithms. The notation Sign($\cdot$) represents the TTP's signature algorithm. The TTP shares with each sensor j a *long-term key* ($K_{LT}^{(j)}$), which is a symmetric key preloaded in the sensors. Each sensor holds also a *data encryption key* ($DEK^{(j)}$), which is a symmetric key used to encrypt the sensed data. For each piece of sensed data, the sensor computes a new data encryption key as a one-way hash of the old one, and the old one is securely destroyed. Each user u holds a decryption key $DK^{(u)}$, and a public/private key pair ($K_{pub}^{(u)}, K_{priv}^{(u)}$) through which (s)he can receive confidential messages by asymmetric encryption. The cloud server maintains a database of re-encryption key histories and a database of decryption key components that allows for proxy re-encryption. In addition it stores *encrypted sensed data*, *sensor key material*, and *ABE-sealed keys.* The encrypted sensed data is the sensed data produced by the sensors and encrypted with the data encryption key. The sensor key material is a set of cryptographic quantities needed by the sensors to encrypt sensed data. The ABE-sealed keys are the data encryption keys encrypted with ABE. Both the sensor key material and the ABE-sealed keys are produced by the TTP. To decrypt a piece of encrypted sensed data, users must first decrypt the corresponding ABE-sealed key thus retrieving the data encryption key, and then decrypt the sensed data with it.

The universe of the attributes $\mathcal{U}$ is logically partitioned into two subsets: $\mathcal{U}_{\mathcal{R}}$ and $\mathcal{U}_{\mathcal{X}}$. The $\mathcal{U}_{\mathcal{R}}$ subset includes the attributes that represent road segments. Such attributes allow us to restrict users to access data sensed only from some paths or zones of the city, specified as a set of road segments (*authorized road segments*). $\mathcal{U}_{\mathcal{X}}$ includes the attributes that represent time. Such attributes allow us to restrict users to access data sensed only within a specific time period (*validity period*). The maximum resolution with which ABE-Cities can represent validity periods is a system parameter that we call *time unit.* Setting a too short time unit (for example, one second) could be infeasible because the TTP should produce a large set of ABE-sealed keys within a time unit (see Section 4.1). A reasonable choice for the time unit is one day.

Each ABE-sealed key relative to the sensor j ($ASK^{(j)}$) is labeled with the encryption attributes $\gamma^{(j)}$, which are logically partitioned into two subsets: $\gamma_{\mathcal{R}}^{(j)}$ and $\gamma_{\mathcal{X}}^{(j)}$. The attributes in $\gamma_{\mathcal{R}}^{(j)}$ identify the road segment in which the sensor is deployed, while the attributes in $\gamma_{\mathcal{X}}^{(j)}$ identify the time unit during which data has been sensed (*data production time unit*). Similarly, the access policy $\mathcal{T}$ embedded in each decryption key is logically partitioned into two subtrees: $\mathcal{T}_{\mathcal{R}}$ and $\mathcal{T}_{\mathcal{X}}$ (Figure 2).

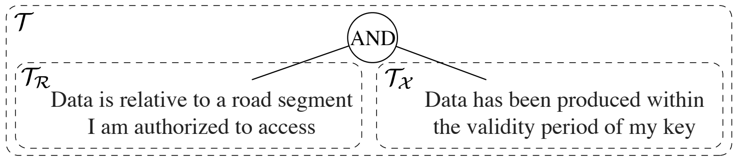

$\mathcal{T}$
AND
$\mathcal{T}_{\mathcal{R}}$
Data is relative to a road segment I am authorized to access
$\mathcal{T}_{\mathcal{X}}$
Data has been produced within the validity period of my key

Figure 2: Example of access policy for a user's decryption key.

The two subtrees are operands of an AND operator ($\mathcal{T} = \mathcal{T}_{\mathcal{R}} \wedge \mathcal{T}_{\mathcal{X}}$), which is the root of the access policy. $\mathcal{T}_{\mathcal{R}}$ identifies the authorized road segments of the decryption key, while $\mathcal{T}_{\mathcal{X}}$ identifies its validity period. In order to decrypt an ABE-sealed key, both the subtrees of the user's decryption key must be satisfied. In the traffic monitoring example, the user can be authorized to access only videos produced after his/her subscription to the service starts, and before it expires. The maximum resolution with which the validity periods can be represented is the time unit.

To implement proxy re-encryption, we use the whole attribute set $\mathcal{U}_{\mathcal{X}}$ in place of the dummy attribute used in the Yu et al.'s PRE scheme [26]. This allows us to save a component for each decryption key and an ABE ciphertext component for each ABE-sealed key. The attribute set $\mathcal{U}_{\mathcal{X}}$ is a good replacer for the dummy attribute because: (1) at least one attribute in $\mathcal{U}_{\mathcal{X}}$ is ANDed at the root of the access policy of all the decryption keys; (2) at least one attribute in $\mathcal{U}_{\mathcal{X}}$ is an encryption attribute of all the ABE-sealed keys; and (3) the attributes in $\mathcal{U}_{\mathcal{X}}$ are never updated to a new version (see Section 4.1). The cloud server is provided with all the decryption key components except those relative to the attributes in $\mathcal{U}_{\mathcal{X}}$.

4.1. System Procedures

In the following, we will implicitly consider the ABE ciphertext components e_i, the decryption key components dk_i, and the re-encryption keys rk_i as always accompanied by the information about their version.

4.1.1. Setup Procedure

This procedure initializes the system. The TTP decides the universe of the attributes $\mathcal{U}$ basing on the street network of the city and on the maximum system lifetime (see Section 5). Then, it executes the Setup primitive which takes the universe of the attributes and produces the master key and the public parameters. The TTP keeps the master key secret and initializes an empty re-encryption key history for each attribute in $\mathcal{U}_{\mathcal{R}}$ in the cloud server.

4.1.2. Key Distribution Procedure

This procedure provides a user u with a decryption key $DK^{(u)}$. It is executed when a new user joins the system, as well as when an old user needs to renew his/her decryption key because it has expired or has been compromised. The TTP first defines the access policy for the new user u. Which access policy assigning to each user strictly depends on the application. In the traffic monitoring example, users can be authorized to view videos from short or long paths, depending on whether they pay low or high dues. Once the TTP has defined the user's access policy $\mathcal{T}^{(u)}$, it creates the user's decryption key by executing:

$$DK^{(u)} = \text{KeyGen}(MK, \mathcal{T}^{(u)}).$$

The TTP encrypts the message $(DK^{(u)}, ts, \text{Sign}(DK^{(u)}, ts))$, where ts is a timestamp, with the user's public key $K_{pub}^{(u)}$ and gives it to the user either directly or through the cloud server. The user verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the user accepts $\mathit{DK}^{(u)}$ as his/her decryption key. In the meanwhile, the TTP sends the message $(u, dk_{i\in\lambda^{(u)}\cap\mathcal{U}_\mathcal{R}}, ts, \mathrm{Sign}(u, dk_{i\in\lambda^{(u)}\cap\mathcal{U}_\mathcal{R}}, ts))$ to the cloud server, where $\lambda^{(u)}$ are the access policy attributes of the user. The cloud server verifies the TTP signature to be valid, and the timestamp to be fresh. If everything is correct, the cloud server stores the decryption key components $dk_{i\in\lambda^{(u)}\cap\mathcal{U}_\mathcal{R}}$.

4.1.3. Seal Procedure

This procedure is executed at the beginning of every time unit. For each sensor j, the TTP randomly generates a data encryption key $\mathit{DEK}^{(j)}$ and encrypts it by executing:

$$\mathit{ASK}^{(j)} = \mathrm{Encrypt}(\mathit{DEK}^{(j)}, \gamma^{(j)}, Y, T_{i\in\gamma^{(j)}}),$$

thus obtaining the ABE-sealed key for the sensor j for the current time unit. The TTP sends the ABE-sealed key to the cloud server, which stores it. Also, for each sensor j the TTP encrypts the message $(\mathit{DEK}^{(j)}, ts, \mathrm{MAC}_{K_{\mathit{LT}}^{(j)}}(\mathit{DEK}^{(j)}, ts))$ with the long-term key $K_{\mathit{LT}}^{(j)}$, where ts is a timestamp and $\mathrm{MAC}_{K_{\mathit{LT}}^{(j)}}(\cdot)$ denotes a message authentication code keyed by $K_{\mathit{LT}}^{(j)}$. Such an encrypted message constitutes the sensor key material for the sensor j for the current time unit. The TTP sends the sensor key material for each sensor to the cloud server, which stores it. Each sensor j retrieves its sensor key material from the cloud server and verifies the message authentication code to be valid and the timestamp to be fresh. If everything is correct, the sensor accepts $\mathit{DEK}^{(j)}$ as its current data encryption key and initializes a *data encryption counter* ($c^{(j)}$) to zero. The sensor uses the data encryption counter to keep track of how many times it renews the data encryption key in the current time unit.

Note that the seal procedure is potentially a long operation for the TTP, since it requires the TTP to execute an Encrypt primitive for each sensor. For this reason, setting a too short time unit (e.g., one second) could be infeasible. A reasonable choice for the time unit is one day.

4.1.4. Data Production Procedure

This procedure is executed every time a sensor j senses a new piece of data. First, the sensor encrypts the sensed data $SD^{(j)}$ with its current data encryption key $\mathit{DEK}^{(j)}$. The result, together with the current data encryption counter $c^{(j)}$, constitutes an encrypted sensed data $\mathit{ESD}^{(j)}$. The sensor sends the encrypted sensed data to the cloud server, which stores it. Then, the sensor computes a new data encryption key as a one-way hash of the old one: $\mathit{DEK}^{(j)} \leftarrow \mathrm{H}(\mathit{DEK}^{(j)})$. Finally, the sensor securely destroys the old data encryption key and increments its data encryption counter. This key renewal method prevents an adversary who compromises a sensor from decrypting old sensed data.

Note that sensors perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices such as battery-powered motes.

4.1.5. Data Consumption Procedure

This procedure is executed each time a user u wants to access a piece of data sensed by the sensor j. The user first sends a data request to the cloud server specifying which sensed data (s)he wants to access. On receiving the data request, the cloud server checks whether some of the stored decryption key components are out of date by checking their versions. For each out-of-date component dk_i, the cloud server updates it by executing:

$$dk_i' = \text{UpdateDK}(i, dk_i, rk_i),$$

and it provides dk_i' to the user. If the decryption key component was outdated of more than one version, then the cloud server will execute the UpdateDK primitive multiple times, each with different re-encryption keys from the re-encryption key history RKH_i. After that, the cloud server checks whether some of the ABE ciphertext components of the ABE-sealed key $\mathit{ASK}^{(j)}$ relative to the sensed data requested by the user are out of date. For each out-of-date component e_i, the cloud server updates it by executing:

$$e_i' = \text{UpdateE}(i, e_i, rk_i),$$

and it replaces e_i with the updated version. If the ABE ciphertext component was outdated of more than one version, then the cloud server will execute the UpdateE primitive multiple times, each with different re-encryption keys from the re-encryption key history RKH_i. Finally, the cloud server provides the user with the encrypted sensed data $\mathit{ESD}^{(j)}$, and the ABE-sealed key $\mathit{ASK}^{(j)}$ relative to the time unit during which the requested data was sensed. On receiving this, the user retrieves the data encryption key by executing:

$$\mathit{DEK}^{(j)} = \mathrm{H}^{c^{(j)}}\left(\text{Decrypt}(\mathit{ASK}^{(j)}, \mathit{DK}^{(u)})\right),$$

where $\mathrm{H}^n(\cdot)$ denotes the one-way hash applied n times. The user finally uses the data encryption key $\mathit{DEK}^{(j)}$ to decrypt the sensed data. Of course, if the user is not authorized to access the data, then the Decrypt primitive will return $\perp$, so s(he) will not able to retrieve the data encryption key.

4.1.6. Key Revocation Procedure

Whenever a user u's decryption key must be revoked, for example when his/her key is compromised, the system executes the key revocation procedure to make the decryption key unable to decrypt any ABE-sealed key.

At first, the TTP determines the updatee set $\mu^{(u)} \subseteq \lambda^{(u)}$, i.e., a set of attributes without which the access policy will never be satisfied. The TTP must update such attributes in order to revoke the decryption key. In our system, the updatee set $\mu^{(u)}$ is formed by the attributes $\lambda^{(u)} \cap \mathcal{U}_\mathcal{R}$. This is because we use the whole attribute set $\mathcal{U}_\mathcal{X}$ in place of the dummy attribute used in the Yu et al.'s PRE scheme [26]. For each attribute $i \in \mu^{(u)}$, the TTP updates the related quantities by executing:

$$(t_i', T_i', rk_i) = \text{UpdateAttribute}(i, \mathit{MK}),$$

and it replaces t_i and T_i with the updated versions. Then, the TTP sends the message $(u, rk_{i,\forall i \in \mu^{(u)}}, ts, \mathrm{Sign}(u, rk_{i,\forall i \in \mu^{(u)}}, ts))$ to the cloud server, where ts is a timestamp. The cloud server verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the cloud server adds each re-encryption key rk_i to the proper RKH_i and discards all the decryption key components $dk_{i \in \lambda^{(u)} \cap \mathcal{U}_{\mathcal{R}}}$ related to the revoked user u. Finally, the TTP executes a seal procedure restricted to those sensors whose at least one ABE ciphertext attribute is in $\mu^{(u)}$, i.e., for those sensors which produce sensed data that the revoked key was authorized to decrypt. We refer to this set of sensors as *affected sensors*, meaning that they are "affected" by a key revocation. The decryption key is now revoked and is not anymore capable of decrypting any ABE-sealed key.

Note that the revocation of a decryption key is a burdensome operation for ABE systems, and it usually causes side effects. Indeed, the decryption keys of all the users sharing at least one attribute in $\mu^{(u)}$ must be updated in order to decrypt new ABE-sealed keys. We refer to this set of users as *affected users*, meaning that they are "affected" by a revocation of the key of another user. A decryption key update is carried out by the cloud server, as described in the data consumption procedure. Another side effect of the revocation of a decryption key is the necessity of generating new ABE-sealed keys with the newest version of the attributes in order to make the decryption key ineffective to decrypt any ABE-sealed key. The generation of the new ABE-sealed keys is carried out by the TTP by means of a seal procedure for the affected sensors, as described earlier.

4.2. Adversary Model(s) and Security Analysis

We distinguish three adversaries differing on their capabilities: (i) someone capable of eavesdropping and injecting traffic between the sensors, the cloud server, and the TTP; (ii) someone capable of compromising one or more sensors; and (iii) someone capable of carrying out a data exfiltration by exploiting some cloud server vulnerability. All these adversaries aim at accessing sensed data without authorization.

The adversary capable of eavesdropping and injecting traffic could either try to steal decryption keys, or inject malicious sensor key material to make the sensors use a comprom ised data encryption key. However, none of these tactics is viable. Stealing decryption keys is infeasible since the TTP sends them to new users in encrypted form. Injecting malicious sensor key material to a sensor is infeasible too, since it is authenticated with the long-term key shared between the sensor and the TTP.

The adversary capable of compromising a sensor can compromise the long-term key and the current data encryption key. However, he can only read sensed data produced by the sensor *after* the compromise, but not that produced before. Indeed, each sensor renews the data encryption key at each new piece of produced data, and the old data encryption key is securely destroyed. The new data encryption key is computed as a one-way hash of the old one, thus it is infeasible to recover the old data encryption keys.

The adversary capable of performing data exfiltration can steal all the encrypted sensed data, the ABE-sealed keys, the sensor key material, and the re-encryption keys. Nevertheless, he cannot access sensed data because it is encrypted. If he owns a decryption key, then he can access only the sensed data that such a decryption key is authorized to decrypt. If the decryption key has been revoked, the adversary can update it by means of the re-encryption keys. Then, he can use such an updated decryption key to decrypt sensed data, even if it has been already re-encrypted by the cloud server. In other words, he can "rollback" the key revocation. As a mitigation measure, the cloud server may store the re-encryption keys in a protected memory space, accessible only by high-privileged processes. On the other hand, sensor key material, encrypted sensed data, and ABE-sealed keys can be accessible by low-privileged processes. As a consequence of this, the process that communicates with sensors and users could run with low privileges, whereas the process that communicates with the TTP should run with high privileges. Supposing that the adversary exploits the former process, which is the more exposed one, he cannot access the re-encryption keys unless he performs privilege escalation. Another mitigation measure is to securely delete old re-encryption keys. Before deleting a re-encryption key, the cloud server must update all the ciphertexts and the decryption key components that use the related attribute. Of course, this can be done without involving the users.

Note that we are interested only in adversaries that aim at illegally access sensed data. Other kinds of adversaries, like someone that injects bogus sensed data into the system or someone that performs a denial of service, fall outside the scope of the paper. In case that the system must defend also against such adversaries, additional defense techniques must be implemented. An adversary that injects bogus sensed data can be counteracted by data authentication mechanisms, for example by accompanying sensed data with Message Authentication Codes (MACs). An adversary that performs a denial of service can be counteracted by IDS/IPS techniques.

5. Universe of the Attributes and Access Policies

In this section we introduce different attribute representations of the street network, resulting in different system performance especially in terms of affected users and *key size*, i.e., the space needed to store a decryption key.

5.1. Road Segments Representation: Universe Subset $\mathcal{U}_{\mathcal{R}}$

5.1.1. Basic Representation

A basic and naive representation consists in mapping one road segment onto one attribute. In this case, the $\gamma_{\mathcal{R}}^{(j)}$ set of each ABE-sealed key is formed by just a single attribute, whereas the $\mathcal{T}_{\mathcal{R}}$ subtree of a decryption key is an OR between many attributes, each one representing a road segment the user is authorized to monitor. We will refer to this implementation as the *basic representation.*

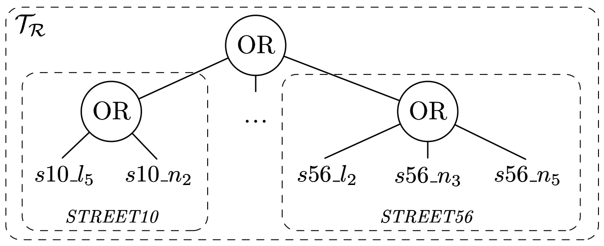


Figure 3: Example of $\mathcal{T}_\mathcal{R}$ subtree of the key access policy with segment-tree representation.

5.1.2. Segment-Tree Representation

The basic representation is very simple to implement, but it has numerous drawbacks. First of all, the number of access policy attributes of a decryption key, and thus the key size, grows linearly with the number of authorized road segments. Moreover, the revocation of a decryption key affects every user whose decryption key is authorized to access *any* road segment that the revoked key was authorized to access. For example, the revocation of a key authorized to access an entire street will affect all the users authorized to access any subset of road segments of that street. Indeed, by construction, in the basic representation, each road segment is mapped onto one attribute. This causes many affected users for each key revocation.

Segment trees can help us to reduce the affected users. In the following, we introduce a representation based on segment trees which aims at limiting the number of users affected by a key revocation, thus making the key revocation more efficient. Let us consider a generic street with ρ road segments, and let us denote the road segments of that street with the identifiers from 1 to ρ. We build a segment tree in which the leaves are the road segment identifiers. Each node of the segment tree is represented by an attribute. The $\gamma_\mathcal{R}^{(j)}$ set of each ABE-sealed key is formed by the point representation set of the sensor j's road segment, and it contains $\mathcal{O}(\log \rho)$ attributes. We use a point-in-interval access policy to authorize the user to access a subset of consecutive road segments of the street. The complete $\mathcal{T}_\mathcal{R}$ subtree of the access policy will be an OR between multiple point-in-interval access policies, one for each street the user is authorized to access. Figure 3 shows a graphic example of this. The key is authorized to access consecutive road segments belonging to two streets, namely *STREET10* and *STREET56.* The subtree $s10\_l_5 \lor s10\_n_2$ is a point-in-interval access policy which allows the user to access a set of consecutive road segments of the street *STREET10.* We will refer to this implementation as the *segment-tree representation.*

Note that, with respect to the basic representation, the TTP is required of a little more effort to produce the ABE-sealed keys since the number of the ABE ciphertext attributes for each street is $\mathcal{O}(\log \rho)$ instead of 1, and the complexity of the Encrypt primitive depends on such a number. However, the segment-tree representation sensibly reduces the number of affected users. Indeed, a user authorized to access a road segment in common with the revoked key is not automatically affected, as it happens in the basic representation. Moreover, the segment-tree representation reduces the key size, because the number of access policy attributes is always less than or equal to the number of authorized road segments.

5.1.3. Attribute-Pool Representation

We introduce a third representation which extends the segment-tree representation to further lower the number of users affected by a key revocation. We replace each attribute i in the universe subset $\mathcal{U}_{\mathcal{R}}$ of the segment-tree representation with a pool of ε attributes $\{i_\omega\}, \forall\omega \in [1, \varepsilon]$ (*replicas*). Each replica i_ω in the pool has the same meaning.

The $\gamma_{\mathcal{R}}^{(j)}$ set of an ABE-sealed key is composed of ε point representation sets and includes $\mathcal{O}(\varepsilon \log \rho)$ attributes, and the size of a decryption key is the same as in the segment-tree representation. Indeed, the structure of the $\mathcal{T}_{\mathcal{R}}$ subtree is equivalent to the one of the segment-tree representation, but each attribute in the subtree is one of the ε replicas. We will refer to this implementation as the *attribute-pool representation.*

To implement this representation, for each replica in $\mathcal{U}_{\mathcal{R}}$ the TTP maintains the number nk_{i_ω} of non-revoked keys using that replica. nk_{i_ω} is incremented when the TTP issues a new decryption key having i_ω as access policy attribute, and it is decremented when a similar key is revoked. When the TTP executes a key distribution procedure, for each attribute which forms the $\mathcal{T}_{\mathcal{R}}$ subtree, it chooses the *least frequently used replica* of the attribute, i.e., $i_{\widetilde{\omega}}$ such that $\widetilde{\omega} = \arg\min_\omega(nk_{i_\omega})$. This is to minimize the number of affected users in case the newly generated key is revoked.

This representation further reduces the number of affected users due to a key revocation with respect to the segment-tree representation. Indeed, two users authorized to access the very same set of road segments may have not attributes in common in their decryption keys if they have different replicas of each attribute. In such a case, if one of them is revoked, the other will not be affected.

5.2. Time Representation: Universe Subset $\mathcal{U}_{\mathcal{X}}$

In ABE-Cities, users can be authorized to access only data sensed within a specific validity period. A “not-before” and a “not-after” times are embedded in each decryption key, in a similar way that X.509 certificates do. Both times are represented with the maximum resolution of the time unit. Let the *maximum system lifetime* be the maximum time of operation of the system. We represent the time units from 1 to χ, where $\chi = \lceil$maximum system lifetime / time unit$\rceil$, by means of a segment tree. A basic representation is possible and consists in mapping each time unit onto one attribute. However, with this representation the number of access policy attributes of a decryption key grows linearly with the number of time units in the validity period. This would produce possible huge-sized keys. To avoid this we adopted a segment-tree representation also for time. The $\gamma_{\mathcal{X}}^{(j)}$ set of each ABE-sealed key is formed by the point representation set of the data production time unit, which is $\mathcal{O}(\log \chi)$ attributes. On the other hand, the validity period of a decryption key is implemented through a point-in-interval access policy, which forms the $\mathcal{T}_{\mathcal{X}}$ subtree and is, again, $\mathcal{O}(\log \chi)$ attributes. The segment-tree representation reduces the key size, because the number of access policy attributes is always less than or equal to the number of time units in the validity period. Note that using an attribute-pool representation would not improve the key revocation performance in this case, since these attributes are never updated.

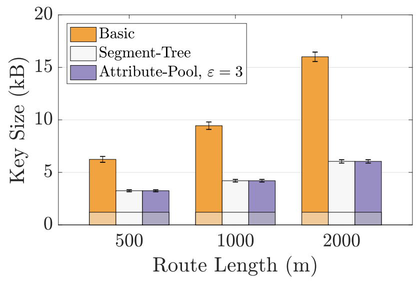
6. Experimental Evaluation

We tested our scheme by simulations, using a street network representing the city of Pisa[3] obtained from OpenStreetMap. We assumed a maximum system lifetime of 100 years and the time unit equal to one day. Then, we considered a dataset of 300 users with randomly chosen 365-day overlapping validity periods. In our simulations, every user is authorized to access a *route* composed of consecutive road segments. A route is characterized by a *route length* L, which is the line-of-sight distance between its source point and its destination point. We chose a source point at random within the map and a destination point at random at a distance L from the source point. Within each scenario we tested, we fixed a value for the route length which is the same for all the users. We generated a decryption key for each user, by using an implementation of the Goyal et al.'s scheme written in the C language [30], which realizes the four ABE primitives described in Section 3.1 with 80-bit of security strength.

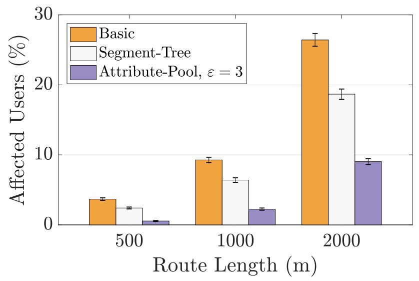
In Figure 4a we show a comparison between the average key sizes for the three road segments representations, with respect to the route length. The lower part of the bars shows the portion of the key size concerning the $\mathcal{T}_{\mathcal{X}}$ subtree. With all the three representations, a user's device, e.g., a smartphone, can easily store a decryption key of a few kilobytes. The cloud server stores only the decryption key components of the $\mathcal{T}_{\mathcal{R}}$ for all the users, and the total size for our dataset of 300 keys is about 1.4 MB using either the segment tree or the attribute-pool representation with route length of 2000 m. The key size of the segment tree and the attribute-pool representations are about a third compared to the basic representation. Nevertheless, the keys are fairly small for all the three representations (< 16 kB in the worst case), so the key size should be affordable for both the users and the cloud server.

By using the same dataset, we revoked each decryption key in turn and measured how many users were affected. In Figure 4b we show the average percentage of affected users by a single key revocation. As expected, the segment tree and the attribute-pool representations show less affected users than the basic representation. This holds for all the route lengths tested. Figure 4c shows the affected users with respect to the number of replicas in the attribute-pool representation.

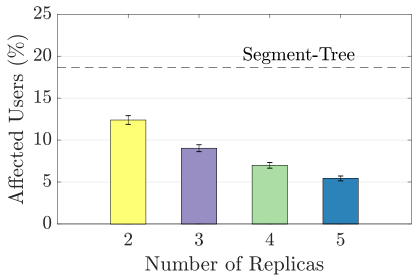
[3]Lat: [43.7052 , 43.7264], Lon: [10.3867 , 10.4266].



(a) Average key size wrt route length.



(b) Average percentage of affected users due to a single key revocation wrt route length.



(c) Average percentage of affected users due to a single key revocation wrt number of replicas in the attribute-pool representation. $L = 2000\,\mathrm{m}$.

Figure 4: Performance evaluation. All the plots show 95 %-confidence intervals in error bars.

It is evident from the figure that the affected users can be reduced even further by increasing the parameter ε. The segment-tree representation and the attribute-pool representation are capable of reducing the number of affected users, and this lightens the load for the cloud server due to a key revocation, which is also distributed over time thanks to the laziness of proxy re-encryption.

In our system, the load on the TTP is mainly determined by the complexity of the Encrypt primitive, which grows linearly with the ABE ciphertext attributes. The TTP executes such primitive once for each sensor at the beginning of the seal procedure. We experimentally evaluated the computational load and the required bandwidth on the TTP for the seal procedure. We used the same street network representing the city of Pisa and a maximum system lifetime of 100 years. We assumed 100 sensors deployed randomly on the street network. We used the attribute-pool representation with $\varepsilon = 5$, which represents the worst considered case in terms of ABE ciphertext attributes, and thus the worst case for the computational load and the required bandwidth on the TTP. We define the *key sealing time* as the time needed to generate all the ABE-sealed keys at the beginning of the seal procedure. We observed a key

	Pisa	Houston	Beijing
short route	500 m	1500 m	5000 m
medium route	1000 m	3000 m	10 000 m
long route	2000 m	6000 m	20 000 m

Table 1: Route lengths.

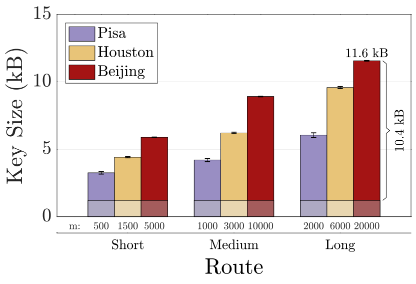
sealing time of 5.4 seconds, and the total size of the ABE-sealed keys was about 550 kB. Note that the seal procedure is executed within the key revocation procedure too, so the key sealing time represents also the minimum time that the TTP takes to revoke a decryption key. We evaluated the key sealing time on a desktop computer equipped with 16 GB of RAM, an Intel® Core™ i5-6600 CPU, and running Ubuntu 16.04.3 LTS 64-bit operating system. From our tests, the ABE ciphertext attributes γ of the average ABE-sealed key are about 39, of which about 21 those of $\gamma_{\mathcal{R}}$. In other words, the TTP is asked to carry out a task of a few seconds and then send a few kilobytes of data to the cloud server every day. This load should be affordable by the average TTP.

6.1. Experimental Evaluation on Large Cities

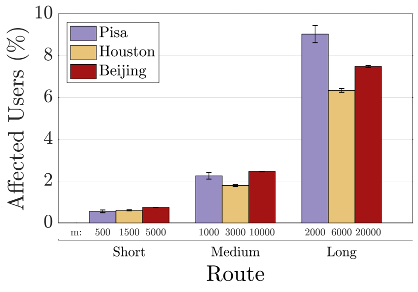
In order to prove the scalability of our system, we performed the same simulations on two large cities, namely Houston[4] and Beijing[5]. We illustrate the effects of revocation in terms of affected users, and we report results which show the average key size in these new scenarios as the length of the routes varies. The area of these cities is about 10 and 100 times the area of Pisa, respectively. Accordingly, we considered proportional datasets of users for Pisa (300 users), Houston (3000 users), and Beijing (30 000 users). Moreover, we considered route lengths roughly proportional to the square root of the city area. For each city, we thus defined "short route", "medium route", and "long route" a route whose length is as specified by Table 1. Again, we assumed a maximum system lifetime of 100 years, a time unit equal to one day, and 365-day validity periods. As we outlined in Section 6, the attribute-pool representation guarantees the best performance with regards to affected users. Therefore, simulations in this section take into consideration only such a representation with $\varepsilon = 3$.

In Figure 5a we show the average key size for the three route lengths tested on each city. The lower part of the bars shows the portion of the key size concerning the $\mathcal{T}_{\mathcal{X}}$ subtree. Of course, for each city, the key size increases with the route length since the set of authorized road segments becomes larger. However, the graph in Figure 5a reveals that the key size is influenced by distinctive features of each city's street network. In other words, street network properties, such as the average road segment length, the average number of road segments

[4]Lat: [29.7171 , 29.7975], Lon: [−95.4145 , −95.3208].
[5]Lat: [39.7705 , 40.0271], Lon: [116.2251 , 116.5581].



(a) Average key size wrt route length.



(b) Average percentage of affected users due to a single key revocation wrt route length.

Figure 5: Performance evaluation on large cities.

within a street, etc., can impact on the key size. Among these properties, we found the *average road segment length* (Table 2) to be the most influential in the outcomes.

Pisa	Houston	Beijing
22.4 m	34.6 m	66.4 m

Table 2: Average road segment length.

In particular, longer road segments produce smaller keys, since the same route length is composed of less road segments. For example, with reference to Figure 5a, the medium route in Beijing produces a key smaller than the long route in Houston, even if the former is longer than the latter (10 000 m versus 6000 m). The reason is that in Beijing the average road segment is 66.4 m long, whereas in Houston it is 34.6 m long. As a consequence, a route in Beijing is composed of less road segments than a route of the same length in Houston, thus resulting in a smaller key. The average road segment length is an indirect measurement of the "density" of the street network since the road segment is basically the distance between two crossroads, so the shorter the distance is, the higher the number of crossroads in the same area. To sum up, small but "dense" cities like Houston results in larger keys with respect to big but "sparse" cities like Beijing.

From Figure 5a it can be seen that, even in the worst case considered, i.e., Beijing with routes of 20 000 m, the average key is about 11.6 kB, which should be affordable for the majority of the users. Such a small average key is also advantageous to the cloud server, which must store all the key components associated with the $\mathcal{T}_{\mathcal{R}}$ subtree for all the users. Considering that the average size of the $\mathcal{T}_{\mathcal{R}}$ subtree key components is about 10.4 kB, and that the Beijing scenario counts 30 000 users, the cloud server must dedicate only 311 MB for the decryption key components, which again should be affordable. This corroborates the scalability of ABE-Cities in terms of key size.

In Figure 5b we show the average percentage of affected users due to a revocation. As expected, this percentage grows with the route length in all the three cities. The reason is that longer routes are more likely to go through the same roads, which in turn makes collisions between users more probable. With many user collisions, a revocation of a user causes a high number of affected users. Interestingly, the small city of Pisa scales particularly bad in terms of affected users as the route length increases. The reason is attributable to the medieval origins of Pisa, whose street network tends to canalize the traffic on few main roads, instead of spreading it on the whole city. This makes collisions between users extremely probable as the route grows in length. Nonetheless, for each city and route length tested, the average percentage of affected users never exceeds 10 %. The worst scenario in terms of absolute number of users affected by a revocation is Beijing with long routes, which gives $7.48\,\% \cdot 30\,000 = 2244$ affected users. This number of affected users should be bearable by the average cloud server, also thanks to the laziness of the mechanism. Indeed, considering that the cloud server updates the key components only upon a data request by the affected user, its computational load is largely spread over time. On the contrary, the computational load of the TTP is sensibly affected by the size of the city. We will thoroughly examine this aspect in Section 8.

7. Advanced ABE-Cities

In the ABE-Cities scheme described in Section 4, the key sealing time grows roughly linearly with the sensors deployed in the city. This is because the TTP executes the Encrypt primitive once for each sensor. Depending on the city size, the number of sensors in the city can be large, and this might overload the TTP. Moreover, the TTP may be a single point of failure of the whole system. If the TTP misses or delays the execution of a seal procedure for some reason (e.g., a technical failure), the sensors cannot upload data to the cloud server any longer. Indeed, they cannot execute the data production procedure since they lack the new sensor key material. In this section we introduce a more advanced scheme which reduces the computational load of the TTP and mitigates the effects of TTP downtime, resulting in a briefer seal procedure.

The advanced ABE-Cities scheme leverages full-resource *gateways*, often deployed in IoT systems, through which the sensors access the Internet. Gateways can execute the Encrypt primitive on behalf of the TTP and in parallel with it, thus reducing the execution time of the seal procedure. This raises the problem of distributing authentic copies of many public parameters to a potentially large set of gateways, which we address by using Merkle trees.

A *Merkle tree* is a binary tree used to efficiently verify the integrity of large data sets or portions of them. Assume that a large data set is partitioned into *n data blocks*. Each data block corresponds to a leaf in the Merkle tree. The label of each leaf (*leaf label*) is a one-way hash of the corresponding data block. The label of each non-leaf node (*non-leaf label*) is a one-way hash of the concatenation of its child nodes' labels. The root label is called *top hash*

(t_H). The integrity of the top hash implies, of course, the integrity of all the data blocks. To verify the integrity of a subset of data blocks, it is sufficient to retrieve the labels of the minimal set of nodes covering the other data blocks (*covering hash set*, $\overline{\eta}$), and a proof of integrity of the top hash, which can be for example a digital signature on it. In the average case, the covering hash set is only $\mathcal{O}(\log n)$, thus the amount of information required to verify the integrity is little compared the complete set of data blocks. Figure 6 shows an example of a Merkle tree.

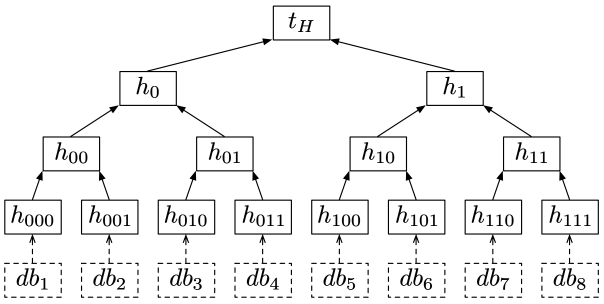


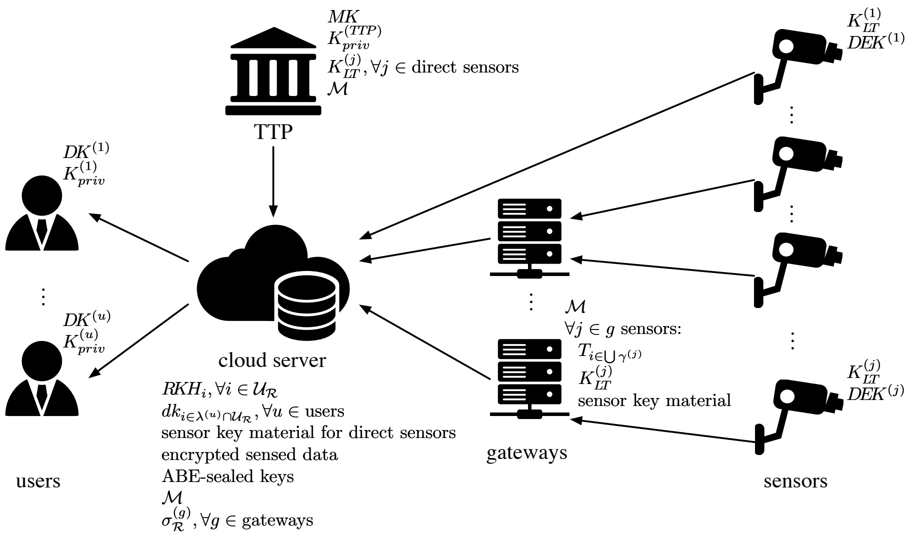
Figure 6: Example of Merkle tree with eight data blocks.

With reference to such an example, if we want to verify the integrity of db_1, db_2 and db_4, we just need to retrieve such data blocks, the set $\overline{\eta} = \{h_{010}, h_1\}$, and the proof of integrity of t_H.

The architecture of the advanced ABE-Cities scheme is shown in Figure 7. We assume that a sensor can access the cloud server either through a gateway (*networked sensor*), for example through a Wi-Fi connection, or directly (*direct sensor*), for example through a Narrowband IoT connection. A sensor j shares a long-term key $K_{LT}^{(j)}$ either with its gateway, in the case it is a networked sensor, or the TTP, in the case it is a direct sensor. Since gateways execute the Encrypt primitive, they need to receive authentic copies of the public parameters from the TTP. The TTP uses a Merkle tree to transmit different subsets of public parameters to different gateways in an authenticated manner. This permits the gateways to verify the public parameters authenticity by receiving only few data. Each gateway must receive the public parameters corresponding to the union of the attributes $\gamma_{\mathcal{R}}^{(j)}$ identifying the road segments of its networked sensors. We call such a set $\sigma_{\mathcal{R}}^{(g)} = \bigcup \gamma_{\mathcal{R}}^{(j)}$ for all the networked sensors j connected to the gateway g. Also, each gateway must receive all the public parameters in $\mathcal{U}_{\mathcal{X}}$.

7.1. System Procedures

With respect to the procedures of the vanilla ABE-Cities scheme explained in Section 4.1, key distribution, data production, and data consumption procedures remain unchanged. In the following, we describe the new version of setup, seal, and key revocation procedures, and we introduce a new one, namely the gateway join procedure.



MK
$K_{priv}^{(TTP)}$
$K_{LT}^{(j)}, \forall j \in$ direct sensors
$\mathcal{M}$
TTP
$K_{LT}^{(1)}$
$DEK^{(1)}$
$DK^{(1)}$
$K_{priv}^{(1)}$
$DK^{(u)}$
$K_{priv}^{(u)}$
cloud server
$RKH_i, \forall i \in \mathcal{U}_\mathcal{R}$
$dk_{i \in \lambda^{(u)} \cap \mathcal{U}_\mathcal{R}}, \forall u \in$ users
sensor key material for direct sensors
encrypted sensed data
ABE-sealed keys
$\mathcal{M}$
$\sigma_\mathcal{R}^{(g)}, \forall g \in$ gateways
users
$\mathcal{M}$
$\forall j \in g$ sensors:
$T_{i \in \bigcup \gamma^{(j)}}$
$K_{LT}^{(j)}$
sensor key material
gateways
$K_{LT}^{(j)}$
$DEK^{(j)}$
sensors

Figure 7: Advanced ABE-Cities architecture.

7.1.1. Setup Procedure

This procedure initializes the system. After having executed the setup procedure of Section 4.1, the TTP creates a Merkle tree $\mathcal{M}$ where data blocks are the public parameters $T_{i \in \mathcal{U}_\mathcal{R}}$ and signs the top hash t_H. Then, the TTP sends the message $(T_{i \in \mathcal{U}}, ts, \text{Sign}(t_H || T_{i \in \mathcal{U}_\mathcal{X}} || ts))$ to the cloud server, where ts is a timestamp. The cloud server re-constructs the Merkle tree with the received public parameters and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the cloud server accepts $T_{i \in \mathcal{U}}$ as authentic public parameters and stores the $\mathcal{M}$ data structure.

7.1.2. Gateway Join Procedure

This procedure is executed every time a new gateway g joins the system. We assume the gateway knows the public key of the TTP, through which it can verify TTP signatures. The gateway communicates to the TTP which road segments its networked sensors are placed onto. The TTP determines the attribute set $\sigma_\mathcal{R}^{(g)}$ and sends the message $(\sigma_\mathcal{R}^{(g)}, ts, \text{Sign}(t_H || T_{i \in \mathcal{U}_\mathcal{X}} || T_{i \in \sigma_\mathcal{R}^{(g)}} || ts))$ to the cloud server. Then, the cloud server sends the message $(\sigma_\mathcal{R}^{(g)}, T_{i \in \sigma_\mathcal{R}^{(g)}}, \overline{\eta}^{(g)}, T_{i \in \mathcal{U}_\mathcal{X}}, ts,$ $\text{Sign}(t_H || T_{i \in \mathcal{U}_\mathcal{X}} || T_{i \in \sigma_\mathcal{R}^{(g)}} || ts))$ to the gateway, where $\overline{\eta}^{(g)}$ is the covering hash set of the public parameters $T_{i \in \sigma_\mathcal{R}^{(g)}}$ and ts is a timestamp. The gateway re-constructs the Merkle tree $\mathcal{M}$ and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the gateway accepts $T_{i \in \sigma_\mathcal{R}^{(g)}}$ and $T_{i \in \mathcal{U}_\mathcal{X}}$ as authentic public parameters and will use them to generate ABE-sealed keys during the seal procedure.

7.1.3. Seal Procedure

This procedure is executed at the beginning of every time unit. First, The TTP executes the seal procedure of Section 4.1 only for the direct sensors. As for networked sensors, each gateway g randomly generates a data encryption key $\mathit{DEK}^{(j)}$ for each networked sensor j and encrypts it by executing:

$$\mathit{ASK}^{(j)} = \text{Encrypt}(\mathit{DEK}^{(j)}, \gamma^{(j)}, Y, T_{i \in \gamma^{(j)}}),$$

thus obtaining the ABE-sealed key for the networked sensor j for current time unit. Also, the gateway g encrypts with the long-term key $K_{LT}^{(j)}$ the message $(\mathit{DEK}^{(j)}, ts, \text{MAC}_{K_{LT}^{(j)}}(\mathit{DEK}^{(j)}, ts))$, where ts is a timestamp and $\text{MAC}_{K_{LT}^{(j)}}(\cdot)$ denotes a message authentication code keyed by $K_{LT}^{(j)}$. Such an encrypted message constitutes the sensor key material for the networked sensor j for the current time unit. The gateway g forwards the sensor key material to its networked sensors. Finally, the gateway securely destroys the data encryption keys and the sensor key materials. This prevents an adversary who compromises the gateway from decrypting sensed data. Each networked sensor verifies the message authentication code to be valid and the timestamp to be fresh. If everything is correct, the networked sensor accepts $\mathit{DEK}^{(j)}$ as its current data encryption key and initializes the data encryption counter $(c^{(j)})$ to zero.

7.1.4. Key Revocation Procedure

This procedure is executed every time a user u's decryption key must be revoked. First, the key revocation procedure described in Section 4.1 is executed, except for the final seal procedure for the affected sensors. The TTP generates then a new Merkle tree $\mathcal{M}'$ where data blocks are the latest version of the public parameters $T_{i \in \mathcal{U}_{\mathcal{R}}}$. Then, the TTP sends the message $(T_{i \in \mu^{(u)}}, ts, \text{Sign}(t'_H || ts))$ to the cloud server, where $\mu^{(u)}$ is the set of attributes updated during the revocation procedure of the vanilla scheme and ts is a timestamp. The cloud server re-constructs the Merkle tree with the latest version of the public parameters and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the cloud server stores the $\mathcal{M}'$ data structure. Then, the cloud server identifies all the *affected gateways*, which are those gateways whose $\sigma_{\mathcal{R}}^{(g)}$ set includes at least one attribute of the revoked key. These gateways are "affected" by the key revocation because the public parameters they use to produce ABE-sealed keys must be updated. The cloud server sends the message $(T_{i \in \{\sigma_{\mathcal{R}}^{(g)} \cap \mu^{(u)}\}}, \overline{\eta}^{(g)\prime}, ts, \text{Sign}(t'_H || ts))$ to each affected gateway. Each affected gateway g re-constructs the Merkle tree and verifies the TTP signature to be valid and the timestamp to be fresh. If everything is correct, the gateway accepts $T_{i \in \{\sigma_{\mathcal{R}}^{(g)} \cap \mu^{(u)}\}}$ as authentic public parameters. Finally, a seal procedure is executed for all the affected sensors. Note that the Merkle tree allows the cloud server to send to each affected gateway only the public parameters it needs, and the affected gateway to accept the parameters by verifying only one signature.

7.2. Gateway Compromise

The advanced scheme inherits the security properties of the vanilla scheme with respect to the threats examined in Section 4.2. However, since gateways are now part of the system, we have to take into account the possibility of their compromise.

An adversary capable of compromising a gateway can compromise the long-term keys of all its networked sensors. However, the gateway securely destroys the data encryption keys and the sensor key materials during the seal procedure. Hence, the adversary will be able to decrypt only sensed data produced *after* the next execution of the seal procedure.

8. Advanced ABE-Cities Evaluation

In the ABE-Cities scheme described in Section 4, the key sealing time grows roughly linearly with the sensors deployed in the city. Depending on the city size, the number of sensors in the city can be large, and this might overload the TTP. The advanced ABE-Cities scheme presented in Section 7 mitigates this problem by outsourcing part of the computational load to the gateways. In this section we validate this by means of experiments.

We used the same street networks representing the cities of Houston and Beijing, and, for all the simulations performed, we assumed a fixed portion (10 %) of the road segments to be covered by a sensor. The sensors were then partitioned in networked and direct. For each simulation we changed the percentage of networked sensors from 0 to 100 % in steps of 10 %. The road segments covered by sensors and which of these sensors are networked were randomly selected for each simulation. We partitioned each city in 100×100 m tiles, and we assumed to have a gateway at the centre of each tile which is connected to all the networked sensors of the tile, if any. This implies gateways to have about 70 m of transmission range, which is realistic for the WiFi protocol. We used the attribute-pool representation with $\varepsilon = 5$ to represent the street network, a maximum system lifetime of 100 years, and a time unit equal to one day. This scenario fits a participatory sensing application in which sensors are owned by different companies or even private citizens, for example a traffic monitoring application with IP cameras.

Since the TTP and the gateways produce their ABE-sealed keys in parallel, the key sealing time will be the maximum between the time required to the TTP (*TTP key sealing time*) and that required to the gateways (*gateway key sealing time*). The gateway key sealing time will be in turn dominated by the most loaded gateway, which is the last one to complete the generation of the ABE-sealed keys. To measure the key sealing time for a city, we generated the ABE-sealed keys by using an implementation of the Goyal et al.'s scheme written in the C language [30]. In particular, we generated an ABE-sealed key for each direct sensor using a desktop computer which emulates the TTP, and an ABE-sealed key for each networked sensor of the most loaded gateway using a single-board computer which emulates the gateway. The desktop computer that emulates the TTP is equipped with 16 GB of RAM, an Intel® Core™ i5-6600 CPU, and running Ubuntu 16.04.3 LTS 64-bit operating system. The single-board computer that emulates the gateway is a Raspberry Pi 3 Model B+, equipped with off-the-shelf hardware (1 GB SRAM and ARM Cortex-A53 1.4 GHz) running Raspbian Jessie.

Figs. 8a and 8b show the TTP key sealing time and the gateway key sealing time for Houston and Beijing, respectively.

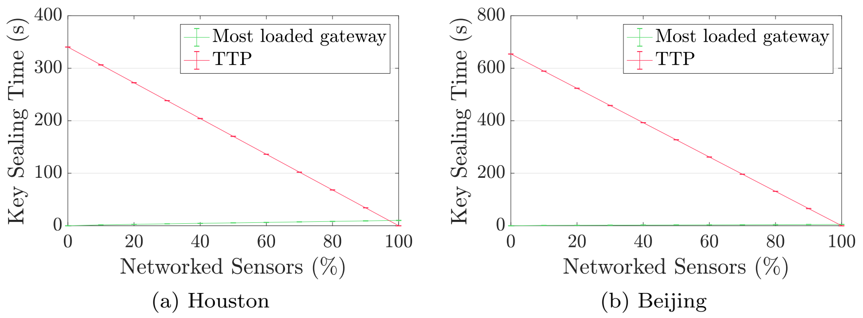


Figure 8: TTP key sealing time and gateway key sealing time wrt the percentage of networked sensors with 100 × 100 m-tile gateways. All the plots show 95 %-confidence intervals in error bars.

Remember that the key sealing time of the whole system will be the maximum between the two. From the graphs we can easily notice that, as the percentage of networked sensors increases, the TTP's load decreases whereas the gateways' load increases, both in a linear fashion. With 0 % of networked sensors, all the load is on the TTP, and we obtain the same performance of the vanilla scheme described in Section 4. The key sealing time of the vanilla scheme is 5 minutes and 40 seconds in Houston, and 10 minutes and 54 seconds in Beijing. With a greater percentage of networked sensors, the key sealing time drops noticeably. This shows that the advanced scheme mitigates the scalability issues of the non-advanced scheme in terms of TTP computational load. The point of intersection of the two curves represents the best case which minimizes the key sealing time. After that, the gateways are overloaded, and they become the bottleneck of the system. The best-case key sealing time is about 9.9 seconds in Houston with 97 % of networked sensors, and 4.9 seconds in Beijing with 99 % of networked sensors. Beijing is larger than Houston but reaches a better performance because it has more gateways, so the computational load is outsourced to a greater extent.

Note that the above scenario requires to have many gateways in the smart city, namely about 8100 in Houston and about 81 000 in Beijing. This is realistic for a participatory sensing application which involves private citizens' WiFi gateways. On the other hand, in case that private gateways could not be involved, deploying such a big number of gateways would be unfeasible. In this case, adopting a long-range and low-bitrate communication technology like

LoRa is a more scalable solution in terms of deployment cost. However, less gateways means that the advanced scheme unburdens the TTP to a lesser extent. To quantify the key sealing time in such a scenario, we run again the simulations on the street networks representing the cities of Houston and Beijing. This time, we partitioned each city in 1×1 km tiles, and we assumed to have a gateway at the centre of each tile which is connected to all the networked sensors of the tile, if any. This implies gateways to have about 700 m of transmission range, which is realistic for the LoRa protocol. This scenario fits applications in which the infrastructure must be cheap and the sensors produce little traffic, for example an air-quality monitoring application in which sensors produce few bytes per hour.

Figs. 9a and 9b show the TTP key sealing time and the gateway key sealing time for Houston and Beijing, respectively.

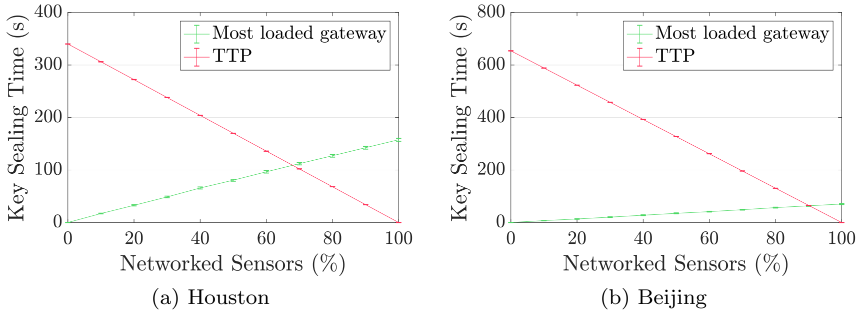


Figure 9: TTP key sealing time and gateway key sealing time wrt the percentage of networked sensors with 1×1 km-tile gateways.

As it can be seen from the graphs, the gateway key sealing time is sensibly higher than the one in the previous WiFi scenario. The reason is that each gateway handles many more networked sensors. As a consequence, the best-case key sealing times are worse too: about 109 seconds in Houston with 68 % of networked sensors, and about 64 seconds in Beijing with 90 % of networked sensors. Nevertheless, the key sealing time decreases noticeably when passing from 0 % of networked sensors (which represents the vanilla ABE-Cities performance) to a greater percentage. This shows that the advanced ABE-Cities scheme scales better than the vanilla one even in the LoRa scenario, and the seal procedure is still briefer.

To sum up, the decentralized nature of the advanced ABE-Cities scheme permits parallelization of the seal procedure and thus its completion in a considerably shorter time period, as outlined by the experimental results. Moreover, it unburdens the TTP of the entire effort required by the seal procedure by outsourcing a significant amount of computation to the gateways, avoids the TTP to be a single point of failure, and provides scalability.

9. Conclusions

In this paper we presented ABE-Cities, an encryption system for smart city applications employing Attribute-Based Encryption. ABE-Cities allows fine-grained access control over the encrypted data stored in the cloud server, and it is secure against traffic eavesdropping, sensors compromise, and data leakage from the cloud server. ABE-Cities senses data from the city and stores it on the cloud server in an encrypted form. Then, it provides users with keys able to decrypt only data sensed from authorized paths or zones of the city. In ABE-Cities, sensors perform only lightweight symmetric-key encryption, thus we can employ constrained sensor devices such as battery powered motes. This makes ABE-Cities suitable for a broader set of IoT smart city applications. ABE-Cities allows us to plan an expiration date for each key, as well as to revoke a given key in an unplanned fashion. We proved that ABE-Cities scales well with the number of users and the number of streets by simulating it with 3000 users on the Houston street network (12 000+ streets), and with 30 000 users on the Beijing street network (30 000+ streets). In addition to the "vanilla" ABE-Cities scheme, we proposed an "advanced" one that leverages the presence of IoT gateways to reduce the TTP computational load. This advanced version is based on Merkle trees, and it takes advantage of nodes acting as sensor gateways to outsource a significant amount of computation from the TTP. We finally validated the advanced scheme's key revocation improved reactivity by testing it on the simulated Houston and Beijing street networks.

Acknowledgments

This work was supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence) and by the project PRA_2018_81 "Wearable sensor systems: personalized analysis and data security in healthcare" funded by the University of Pisa.

References

[1] W. G. Halfond, J. Viegas, A. Orso, et al., A classification of SQL-injection attacks and countermeasures, in: Proceedings of the IEEE International Symposium on Secure Software Engineering, Vol. 1, IEEE, 2006, pp. 13–15.

[2] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom, Spectre attacks: Exploiting speculative execution, arXiv preprint arXiv:1801.01203 (2018).

[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, et al., Meltdown: Reading kernel memory from user space, in: 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 973–990.

[4] S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, Over-encryption: management of access control evolution on outsourced data, in: Proceedings of the 33rd international conference on Very large data bases, VLDB endowment, 2007, pp. 123–134.

[5] L. Coppolino, S. D'Antonio, G. Mazzeo, L. Romano, Cloud security: Emerging threats and current solutions, Computers & Electrical Engineering 59 (2017) 126–140.

[6] A. Sahai, B. Waters, Fuzzy identity-based encryption, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2005, pp. 457–473.

[7] V. Goyal, O. Pandey, A. Sahai, B. Waters, Attribute-based encryption for fine-grained access control of encrypted data, in: Proceedings of the 13th ACM conference on Computer and communications security, ACM, 2006, pp. 89–98.

[8] J. Bethencourt, A. Sahai, B. Waters, Ciphertext-policy attribute-based encryption, in: Security and Privacy, 2007. SP'07. IEEE Symposium on, IEEE, 2007, pp. 321–334.

[9] R. Ostrovsky, A. Sahai, B. Waters, Attribute-based encryption with non-monotonic access structures, in: Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 195–203.

[10] S. Yu, K. Ren, W. Lou, FDAC: Toward fine-grained distributed data access control in wireless sensor networks, IEEE Transactions on Parallel and Distributed Systems 22 (4) (2011) 673–686.

[11] M. Rasori, P. Perazzo, G. Dini, ABE-cities: An attribute-based encryption system for smart cities, in: 2018 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2018, pp. 65–72.

[12] X. Yao, Z. Chen, Y. Tian, A lightweight attribute-based encryption scheme for the internet of things, Future Generation Computer Systems 49 (2015) 104–112.

[13] V. Odelu, A. K. Das, M. K. Khan, K.-K. R. Choo, M. Jo, Expressive CP-ABE scheme for mobile devices in IoT satisfying constant-size keys and ciphertexts, IEEE Access 5 (2017) 3273–3283.

[14] Q. Huang, L. Wang, Y. Yang, DECENT: Secure and fine-grained data access control with policy updating for constrained IoT devices, World Wide Web 21 (1) (2018) 151–167.

[15] M. Ambrosin, A. Anzanpour, M. Conti, T. Dargahi, S. R. Moosavi, A. M. Rahmani, P. Liljeberg, On the feasibility of attribute-based encryption on internet of things devices, IEEE Micro 36 (6) (2016) 25–35.

[16] X. Wang, J. Zhang, E. M. Schooler, M. Ion, Performance evaluation of attribute-based encryption: Toward data privacy in the IoT, in: Communications (ICC), 2014 IEEE International Conference on, IEEE, 2014, pp. 725–730.

[17] N. Oualha, K. T. Nguyen, Lightweight attribute-based encryption for the internet of things, in: Computer Communication and Networks (ICCCN), 2016 25th International Conference on, IEEE, 2016, pp. 1–6.

[18] M. La Manna, P. Perazzo, M. Rasori, G. Dini, fABElous: An attribute-based scheme for industrial internet of things, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 33–38.

[19] B. Girgenti, P. Perazzo, C. Vallati, F. Righetti, G. Dini, G. Anastasi, On the feasibility of attribute-based encryption on constrained IoT devices for smart systems, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2019, pp. 225–232.

[20] A. Lounis, A. Hadjidj, A. Bouabdallah, Y. Challal, Secure and scalable cloud-based architecture for e-health wireless sensor networks, in: Computer communications and networks (ICCCN), 2012 21st international conference on, IEEE, 2012, pp. 1–7.

[21] C. Hu, N. Zhang, H. Li, X. Cheng, X. Liao, Body area network security: a fuzzy attribute-based signcryption scheme, IEEE journal on selected areas in communications 31 (9) (2013) 37–46.

[22] L. Ibraimi, M. Asim, M. Petković, Secure management of personal health records by applying attribute-based encryption, in: Wearable Micro and Nano Technologies for Personalized Health (pHealth), 2009 6th International Workshop on, IEEE, 2009, pp. 71–74.

[23] J. A. Akinyele, M. W. Pagano, M. D. Green, C. U. Lehmann, Z. N. Peterson, A. D. Rubin, Securing electronic medical records using attribute-based encryption on mobile devices, in: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM, 2011, pp. 75–86.

[24] S. Roy, M. Chuah, Secure data retrieval based on ciphertext policy attribute-based encryption (CP-ABE) system for the DTNs, Lehigh CSE Tech. Rep. (2009).

[25] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, D. Starin, Persona: an online social network with user-defined privacy, SIGCOMM Comput. Commun. Rev. 39 (August 2009).

[26] S. Yu, C. Wang, K. Ren, W. Lou, Achieving secure, scalable, and fine-grained data access control in cloud computing, in: Infocom, 2010 proceedings IEEE, IEEE, 2010, pp. 1–9.

[27] G. Wang, Q. Liu, J. Wu, Hierarchical attribute-based encryption for fine-grained access control in cloud storage services, in: Proceedings of the 17th ACM conference on Computer and communications security, ACM, 2010, pp. 735–737.

[28] M. De Berg, M. Van Kreveld, M. Overmars, O. C. Schwarzkopf, Segment trees, in: Computational geometry, Springer, 2000, pp. 231–237.

[29] N. Attrapadung, G. Hanaoka, K. Ogawa, G. Ohtake, H. Watanabe, S. Yamada, Attribute-based encryption for range attributes, in: International Conference on Security and Cryptography for Networks, Springer, 2016, pp. 42–61.

[30] Y. Zheng, kpabe, `https://github.com/altu341com/kpabe`, (Last accessed: 1 April 2019) (2011).